**Web前端工程化方案**

**目 录**

[第一章 前端工程化 5](#_Toc444519286)

[第一节 工程化 5](#_Toc444519287)

[第二节 web前端的发展 5](#_Toc444519288)

[第三节 web前端工程化 5](#_Toc444519289)

[一、 前端工程化的来源 5](#_Toc444519290)

[二、 前端工程化解决问题 6](#_Toc444519291)

[第二章 前端工程化工具及使用 6](#_Toc444519292)

[第一节 环境 6](#_Toc444519293)

[第二节 工具 6](#_Toc444519294)

[一、 NPM 6](#_Toc444519295)

[二、 Bower 7](#_Toc444519296)

[三、 Gulp 7](#_Toc444519297)

[四、 Karma 7](#_Toc444519298)

[五、 Jasmine 7](#_Toc444519299)

[六、 Protractor 7](#_Toc444519300)

[第三章 自动化内容 8](#_Toc444519301)

[第一节 文件合并压缩 8](#_Toc444519302)

[一、 文件合并 8](#_Toc444519303)

[二、 文件压缩 8](#_Toc444519304)

[三、 效率 8](#_Toc444519305)

[第二节 Html拼接 8](#_Toc444519306)

[第三节 sass编译 9](#_Toc444519307)

[第四节 css雪碧 9](#_Toc444519308)

[第五节 图片无损压缩 9](#_Toc444519309)

[第六节 缓存更新 9](#_Toc444519310)

[第七节 js代码质量检测 10](#_Toc444519311)

[第八节 js单元测试 10](#_Toc444519312)

[一、 概述 10](#_Toc444519313)

[二、 单元测试代码 10](#_Toc444519314)

[三、 单元测试结果 10](#_Toc444519315)

[第九节 端对端测试 11](#_Toc444519316)

[一、 概述 11](#_Toc444519317)

[二、 测试代码 11](#_Toc444519318)

[三、 问题 12](#_Toc444519319)

[第四章 工具安装及使用 13](#_Toc444519320)

[第一节 项目目录结构及文件用途 13](#_Toc444519321)

[第二节 工具安装 14](#_Toc444519322)

[第三节 工具使用 14](#_Toc444519323)

[一、 Npm和bower的使用 14](#_Toc444519324)

[二、 Gulp的使用 14](#_Toc444519325)

[三、 单元测试的使用 14](#_Toc444519326)

# 前端工程化

## 工程化

借用苏宁前端架构师(@xufei)的一个总结来说明工程化。

编程技术及生态发展的三个阶段：

* 最初的时候人们忙着补全各种API，代表着他们拥有的东西还很匮乏，需要在语言跟基础设施上继续完善。
* 然后就开始各种模式，标志他们做的东西逐渐变大变复杂，需要更好的组织了。
* 然后就是各类分层MVC，MVP，MVVM之类，可视化开发，自动化测试，团队协同系统等等，说明重视生产效率了，也就是所谓工程化。

## web前端的发展

随着时间的推移，web前端从一开始的切个信息展示的页面，到复杂的pc web app，再到移动端app。前端开发面向的已经不是页面，而是应用了。前端的开发模式也更接近于传统GUI软件，而且由于现在前端领域为了解决日益复杂的web业务需求及体量，越来越多的借鉴了传统客户端的开发经验，导致两者变得越来越趋同。再加上前端一些独特的特性(免安装、增量安装等)，工程上的复杂度有过之而无不及。

## web前端工程化

### 前端工程化的来源

前端三种完全不一样的语言，加上跨浏览器、跨平台，而且前端一直没有统一的规范。导致不同公司，甚至相同公司不同项目的前端规范完全迥异。所以前端无法像java有IDE或者类似Maven的工具用于规范管理代码，前端团队只能利用其它语言编写的工具来分别处理合并，压缩等问题。

Nodejs的发布，使前端工程师脱离了浏览器的束缚。大量前端工具出现在前端生态中，诸如模块规范（CMD、AMD），构建工具（Grunt、Gulp），第三方库管理（nmp、bower）等。前端终于脱离茹毛饮血的时代，进入了工业化时代，利用工程化工具自动化处理很多繁琐的工作。

### 前端工程化解决问题

1. 合理的开发流程及开发规范，包括代码规范、模块化组件化规范等（提高生产力）。
2. 一套自动化代码质量检测方案（提高系统可靠性）。
3. 一套自动化及高度适应性的项目 发布／部署 方案（提高系统的伸缩性及灵活性）。
4. 极致的性能优化，包括减少冗余的接口请求及资源请求、提高缓存命中率等，简言之就是站点的打开及运行速度（更好的用户体验）。

# 前端工程化工具及使用

## 环境

* Node.js：目前前端工程化工具都是基于node.js开发。并且前端工程师能利用node.js快速起页面渲染服务器，利用一些Mock工具脱离后端，进行独立开发。
* Ruby：用于编译sass。

## 工具

### NPM

NPM的全称是Node Package Manager，是一个node.js包管理和分发工具，它的主要功能就是管理node包，包括：安装、卸载、更新、查看、搜索、发布等。在服务器端的node.js中，NPM用于管理第三方包。而web前端只利用NPM管理开发过程用到的工具。

### Bower

Bower是一个客户端技术的软件包管理器，它可用于搜索、安装和卸载如JavaScript、HTML、CSS之类的网络资源。简单说就是前端的maven。

### Gulp

Gulp 是一款基于任务的设计模式的自动化工具，通过插件的配合解决全套前端解决方案，如静态页面压缩、图片压缩、JS合并、SASS同步编译并压缩CSS、服务器控制客户端同步刷新。

Gulp基于node.js的i/o流实现，比grunt有跟高的效率。并且遵循代码优于配置策略，代码更简洁。

### Karma

Karma是一个基于Node.js的JavaScript测试执行过程管理工具（Test Runner）。该工具可用于测试所有主流Web浏览器，也可集成到CI（Continuous integration）工具，也可和其他代码编辑器一起使用。Karma的一个强大特性就是，它可以监控(Watch)文件的变化，然后自行执行，通过console.log显示测试结果。

### Jasmine

Jasmine 是一款 JavaScript 单元测试框架，独立的行为驱动开发框架，语法清晰易懂。

### Protractor

Protractor 是 AngularJS 团队构建的一个端对端的测试运行工具，模拟用户交互，验证web应用的运行状况。”

Protractor使用Jasmine测试框架来定义测试。Protractor为不同的页面交互提供一套健壮的API。

# 自动化内容

## 文件合并压缩

### 文件合并

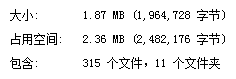
js和css文件合并能有效减少资源请求数量。并且前端当采用了模块和组件开发后，将会产生大量js文件，文件合并就必不可少。

### 文件压缩

文件压缩将去除文件中的注释、空格和回车，并且对js中变量替换为更短的字母组合，可以大大减少文件大小，降低网络资源消耗。

### 效率

压缩效率主要取决于注释数量和变量名长度。 采用某框架的js文件做测试。

合并压缩前：

合并压缩后：C:\Users\Administrator\Desktop\QQ截图20160225171433.png

压缩率为33.5%。

## Html拼接

对html文件进行预处理，将头尾html文件，在发布前就拼接好，减少运行时再jsp拼接的资源消耗。

如果采用组件化开发，也需要该功能拼接声明的各个组件，生成对应的页面。

## sass编译

对sass文件进行编译，生成css文件。

## css雪碧

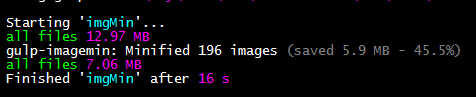
CSS雪碧 即CSS Sprite，是一种CSS图像合并技术，该方法是将小图标和背景图像合并到一张图片上，然后利用css的背景定位来显示需要显示的图片部分。

该功能能通过检查css文件，将其中声明需要sprite的图片合成为一张图片，并且自动修改css文件里的内容，加入background-position属性，正确显示原有图像。

Css sprite能减少资源请求数量。

## 图片无损压缩

通过Photoshop等软件切出的图片会带有很多无关信息，删除这些信息能减少图片大小。下图可见，压缩效率能达到55%。



## 缓存更新

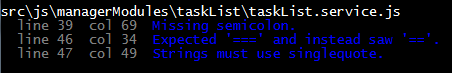
为了提升浏览器打开页面的速度，一般都会让浏览器直接缓存图片或文件，不用每次向服务器请求资源。但版本更新后，浏览器还会使用缓存中旧文件，造成页面错误。

解决缓存更新问题，一般采用更改文件名或增加请求后缀。使用前端构建工具能根据文件内容计算md5，同时修改文件名和引用地址。这样非覆盖更新，还能解决CDN部署不同步的问题。

## js代码质量检测

js作为脚本语言，一直以来都没有官方的规范，导致代码风格各式各样，IDE也难以作错误检查，代码质量无法得到保证。

通过代码质量检查工具，检查代码质量以及找出一些潜在的代码缺陷。使用配置文件统一规范，督促团队成员的代码统一风格。

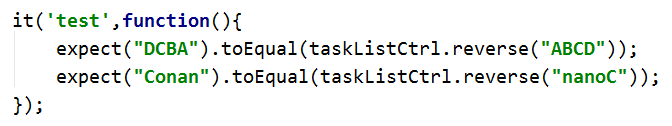


## js单元测试

### 概述

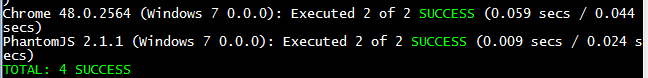
前端业务需求越来越复杂，代码越来越多，为了保证代码质量和产品质量，单元测试就变得必要。由于UI相关的代码做单元测试很麻烦，所以前端单元测试开展一直不顺利。但目前前端已经开始使用mvc框架或采用mvc思想编程，单元测试已经可以实现。

### 单元测试代码

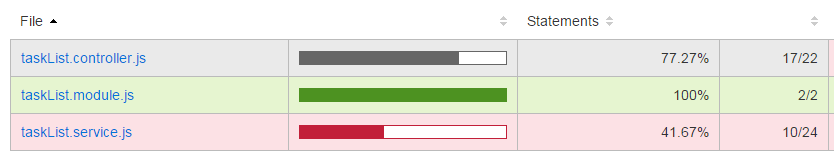


### 单元测试结果

测试能在多种环境中进行，并统一显示结果，保证js代码的跨平台性。目前支持Chrome，ChromeCanary，Firefox，Opera，Safari(only Mac)，IE (only Windows)，PhantomJS（基于webkit内核无ui的浏览器）。



单元测试完毕，还会自动生成每个测试环境覆盖率报告。



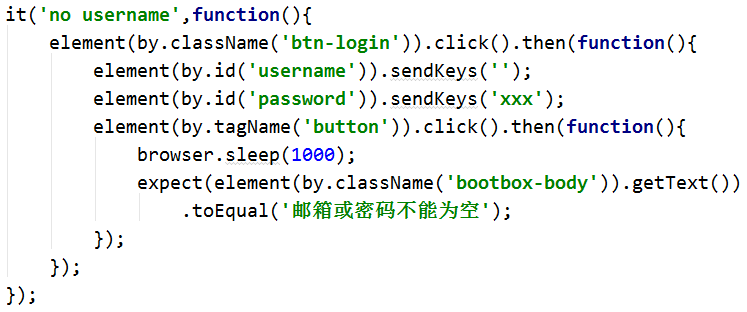
## 端对端测试

### 概述

端对端测试是模拟真实的用户场景，把产品或服务当作一个整体进行验证。换句话说就是把某个功能的UI操作流程写为代码，由测试工具自动执行操作，并检测表现层是否符合预期。

端对端测试不依赖开发环境，可以直接测试线上产品。所以端对端测试更适合测试人员使用，用于优化黑盒测试。

### 测试代码

****

该实例测试登录时，用户名未输入，是否能正确弹出提示。

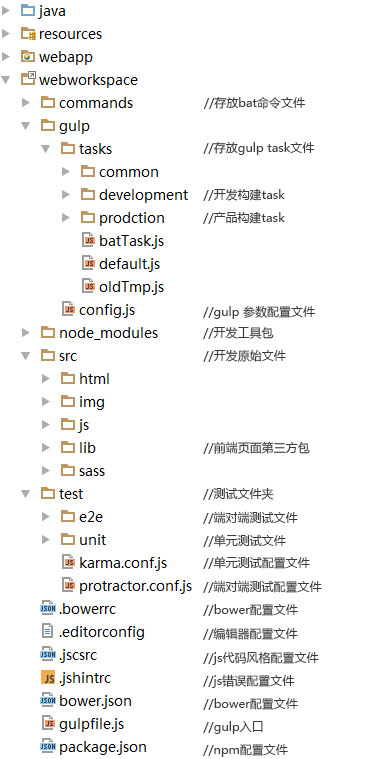
### 问题

写端对端测试代码对人员水平有很高的要求。首先测试代码是用JavaScript写的，并且运行在node.js，所以需要很熟悉JavaScript。其次模拟UI操作需要对html、css有一定的了解。最后还得熟悉protractor和jasmine提供的api。

另外端对端测试依赖Selenium 本地独立服务器，所以环境依赖于node.js和jdk。

# 工具安装及使用

## 项目目录结构及文件用途



## 工具安装

1. Node.js和ruby可以直接在官网下安装包，一键安装。
2. 执行commands文件夹内的install.bat，自动安装开发依赖工具包和页面依赖包。
3. 执行commands文件夹内的protractorInstall.bat，安装端对端工具。

## 工具使用

### Npm和bower的使用

* 使用命令“npm install”安装package.json内的包。
* 使用命令“bower install”安装bower.json内的包。

### Gulp的使用

* 使用命令“gulp”,打在开发环境的包。
* 使用命令“gulp --production”,打在产品环境的包。
* 使用命令“gulp watch”，监听文件是否被修改，若修改则执行对应的开发打包任务。例如某sass文件被修改后，自动执行sass编译任务。
* 可以使用命令“gulp *taskName*”单独执行某个任务。
* 关于gulp的详细使用，请查阅<http://www.gulpjs.com.cn/>。

### 单元测试的使用

* 单元测试已集成进gulp构建流程，测试不通过，将不进行构建。也可以使用“gulp unitTest”或命令“karma start test/karma.conf.js”单独进行单元测试。
* 执行完单元测试后，“test/log”下会生成该次测试覆盖情况。

### 端对端测试的使用

使用命令“protractor test/protractor.conf.js”,启动端对端测试。

# 总结